End-of-Studies Project

Wrecks research with underwater robots

Joris Tillet
Option Robotique, Promotion 2018, ENSTA Bretagne

supervised by
Luc Jaulin

From March to September 2018

Résumé / Abstract

English version below

Ce rapport présente le travail et les résultats obtenus pendant mon projet de fin d'études réalisé au Lab-STICC à l'ENSTA Bretagne, à Brest.

J'ai travaillé sur divers projets pendant les six mois de stage, aussi bien avec du travail d'ingénieur que de chercheur. Ainsi, j'ai automatisé en cap un bateau semi-rigide avec Maël Le Gallic, que nous avons par la suite nommé "Boatbot". J'ai également travaillé sur l'estimation des courants avec un filtre de Kalman, et montré les résultats satisfaisant des simulations. J'ai aussi entamé des recherches sur la localisation bathymétrique par analyse d'intervalles, en vue de localiser un sous-marin sans remonter à la surface mais en connaissant la carte bathymétrique à l'avance. Finalement, j'ai participé à l'intelligence du voilier de l'ENSTA Bretagne, présenté au concours de la WRSC, avec notamment un algorithme de contrôle précis utilisé dans des slaloms serrés.

Mots clés : Boatbot, Estimation des courants, Filtre de Kalman, WRSC, contrôle.

This report presents the work and the results of my end-of-studies project, in the Lab-STICC, in ENSTA Bretagne in Brest in France.

During six months, I worked on various projects, included engineering or research work. So, Maël Le Gallic and I automatized a rigid-hulled inflatable boat, named "Boatbot". I also worked on the estimation of the currents with a Kalman filter, and showed good results in simulations. I carried research on bathymetric localization, with interval analysis, in order to localize a submarine without resurfacing but knowing the bathymetric map. Finally, I helped in the development of the intelligence of the sailboat robot of the ENSTA Bretagne for the contest WRSC. I spent time in a control algorithm for tight slalom.

Keywords : Boatbot, Currents estimation, Kalman filter, WRSC, control.

Acknowledgements

Firstly, I would like to thank my supervisor, Luc Jaulin, who enables me to realize my end-of-studies project. I would also like to thank all the team of Lab-STICC, answering my questions and helping me to integrate the lab, especially Fabrice Le Bars, Benoît Zerr, Thomas Le Mézo, Maël Le Gallic and Auguste Bourgois.

Contents

List of Figures

List of acronyms

AUV	*Autonomous Underwater Vehicle*
DRASSM	*Département de Recherches Archéologiques Subaquatiques et Sous-Marines*
IMU	*Inertial Measurement Unit*
IRSC	*International Robotic Sailing Conference*
RTK	*Real-Time Kinematic*
SBP	*Sub Bottom Profiler*
SLAM	*Simultaneous Localization And Mapping*
ROS	*Robot Operating System*
WRSC	*World Robot Sailing Championship*

Introduction

Wrecks research is a complex task which requires many skills in a range of fields : history, hydrography, marine archeology, etc. However, it has already shown its importance in the understanding of human activities around the seas [10]. Finding a specific wreck is very difficult, and even more for ancient wreck, since there is often a lack of information, and most of the time it is not very accurate and novelize.

That is why the most often, a wreck research requires to lock down a whole area, which can be very spread. This leads to many hours in the sea to scan every place to be sure not to miss the wreck. So the choice of the sensor which will detect the wreck is very tricky. Indeed, the sensor must give a positive detection when there is a wreck, and also do not give too many positive detections because we can not send a diver in too many places. Furthermore, it is more desirable to chose a sensor which scans a large surface in order to reduce the number of kilometers to navigate, and save time and energy.

Finally, we understand that the wrecks research is intricate, and that it requires a long time in the sea to lock down an area, in any case. And it is here that the interest of robotics becomes logical. We can easily imagine a swarm of robots which explore a while spread area very quickly and watched by only one human. Now a wreck research seems less long and difficult.

My internship was divided in different parts. I spent a lot of time in the conception of Boatbot, an automatic rigid-bulled inflatable boat. Only the heading was regulated, but everything has been developed by the Lab-STICC and there were a lot of work. The goal of this robot was to understand what kind of missions must be done to research a wreck, in two dimensions to start. Indeed, the marine place is not easy to lock down and follow lines accurately is quickly intricate, especially when the sea is rough. It was developed in the context of the research of *La Cordelière* [2].

One of the biggest issue on the seas are the currents [4]. Even with an engine, if the currents are not too weak with respect to the boat speed, you can navigate like a crab just because of currents. So, in order to have a correct control, a good idea is to estimate these currents. And it can be done with a Kalman filter [7] ; that is the second part of my job.

At the same time, the lab was assembling a team in order to partake in an autonomous sailboat competition (WRSC[1] [1]), so I joined the team. We worked on the writing of an article about a control of sailboat robots in tight slalom.

Finally, as the final goal is to control underwater vehicles, I also worked on the localization of an AUV[2]. Indeed, as GPS can not be used under the water, to have an accurate localization without regularly resurface is a real issue today. The idea in this part is to use the bathymetry of the area, and with a sonar, to localize the robot by using interval analysis [8] in order to find possible places which fit with the measures.

[1] WRSC : World Robot Sailing Championship
[2] AUV : Autonomous Underwater Vehicle

Chapter 1

Boatbot, an automatic rigid-hulled inflatable boat

The Lab-STICC owns a rigid-hulled inflatable boat, or zodiac, where an electric motor was added in order to turn the wheel automatically, and therefore to control the boat with a computer and without touching the wheel anymore. Now, this zodiac is known under the name of "Boatbot", since it has participated in the research of a wreck, *La Cordelière*, in the *Rade de Brest*, by taking measures automatically.

1.1 *La Cordelière*, the researched wreck

La Cordelière is a french military ship from Brittany. It was armed with 200 cannons, and could allow more than 1,000 crew members [6]. It defended Brittany against the English when they came in the *Rade de Brest* in 1512. It sunk during a terrible battle while it was in a boarding with the *Regent*, an English ship which sunk too at the same time.

Today, the research area is about 100km^2 (see Figure 1.1), and underwater archeology campaigns have been already carried out. The DRASSM[1] is a main actor in these campaigns, and leads the researches. As the ship sunk more than 5 centuries ago, the wreck is probably under about 5 meters of sediments, according to the estimations.

1.2 Mechanical and electronic part

Boatbot is a play on words : it is a robot and a boat. It means that there are enough sensors aboard to navigate alone, without any human help. The only actuator is the electrical engine placed behind the wheel, which has been chosen because it is hidden in the console, so it does not take space aboard (the boat is quite small), and because it can be de-clutched, so anyone can take the control back in case of emergency. Actually, there is always someone aboard, because it is mandatory, and furthermore the speed of the boat is not controlled automatically.

There are two sensors located on a pole : a GPS antenna with its RTK[2] module and an IMU[3]. There is also a splash-proof box aboard with the embedded computer inside, and some power

[1] DRASSM : a French service from the Ministry of Cultural Affairs for underwater archeology.

[2] RTK : Real-Time Kinematic, a satellite navigation technique used to enhance the precision of position data.

[3] IMU : Inertial Measurement Unit, an electronic device using a combination of accelerometers and gyroscopes, and also magnetometers.

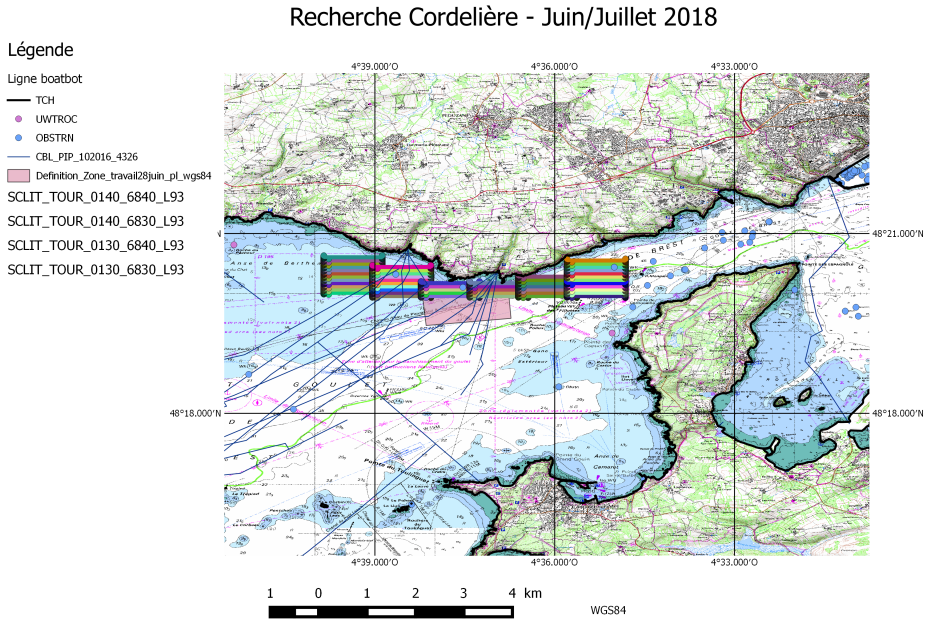


Figure 1.1: Research zone for *La Cordelière*.

electronics. There is also a 4G key in order to have access to the internet (mainly to download RTK corrections), and a motor controller which sends the correct signals to the electric motor. The computer and the electric motor are both powered by the battery of the zodiac, which is itself powered by the engine of the boat.

The last device is a computer for the user, with the interface to watch the state of the boat and launch missions.

With the scope of wrecks research, some sensors can be added to the boat. For instance, we added a mono-beam sonar, a magnetometer or also a SBP[4]. Each of these sensors needs the boat to follow many lines faithfully, and that is why the heading is controlled automatically, since it is too long and heavy for a human and he will probably make errors. Examples of data taken by these sensors (SBP and magnetometer) are presented in Appendix A.

On the Figure 1.2, you can see Boatbot towing a canoe, and the canoe is towing a magnetometer. We used the canoe here to be sure the cable is safe and will not fall in the propeller of the motor.

Concerning the software architecture, we chose to use the middleware ROS[5] [5] to help us to separate the different nodes of our system. So each driver for the sensors or the actuator was in a separated program, and even the intelligence was divided into different nodes which can therefore work at the same time. The communication between the nodes is managed by ROS using topics and services.

[4]SBP : Sub Bottom Profiler, a sediments sounder.
[5]ROS : Robot Operating System.

Figure 1.2: Picture of Boatbot towing a canoe towing a magnetometer.

1.3 Control and line following

The first step of the control of the boat is to succeed in following a given heading. A simple proportional controller is well enough, for the most cases. It means, if we note e the error between the real heading and the desired heading, and u the command, that we have :

$$u = k_p \cdot e = k_p \cdot (\theta_d - \theta_r) \tag{1.1}$$

with θ_d the desired heading and θ_r the real heading, and where k_p is a coefficient used to tune the smooth of the control.

Obviously, this control depends directly on the quality of the sensors, because the real heading and the measured heading are different, most of the time. In our case, with the sensors available on the boat, there are two ways to get the heading. The first one is the IMU, which gives directly where the boat heads. The second one is named the *course over ground*, and it is computed with two successive GPS fixes. They are both interesting because the first one gives the information at very high frequency, and works at any speed of the boat, whereas the second one is accurate only if the speed of the boat is relatively important. The GPS gives information every seconds, which is very slow so it complicates the control. However, it takes into account the currents, which can not be neglected. Indeed, a boat can move like a crab when it navigates in a sea with important currents.

After several tests, the decision of taking the information of heading from the IMU was taken. Indeed, when the boat was turning, as the course over ground was available at very low frequency, it was impossible to know exactly when the heading was the desired one. So there were always unwanted overshoots in the curves.

Then, once the boat succeeded in following correctly a given head, it was already possible to reach waypoints. However, the goal here is to follow a line, which is different. The robot must stay close to the line, it is more important than reach the final waypoint of the line. If we define a line by two waypoints A and B, just to reach A and then reach B will not give a nice route.

This is why an algorithm is proposed here to realize proper line following.

So a vector field is designed in order to give the best heading the robot should have whatever its position. The arctan function is used here because it gives a perpendicular direction to the line when the robot is very far from the line, a parallel direction when the robot is already on the line, and with a nice curve in the middle.

In order to find this algorithm, we need first to compute the angle of the line we want to follow.

If we define the line with two waypoints, $A = (A_x, A_y)$ and $B = (B_x, B_y)$, then we can compute θ, the angle of the line :

$$\theta = \frac{\arccos(B_x - A_x)}{\sqrt{(B_y - A_y)^2 + (B_x - A_x)^2}}. \tag{1.2}$$

The distance of the robot from the line is also important to know, let us note it d. We also note $P = (x, y)$ the position of the robot. Then we have :

$$d = \frac{|\,(B_y - A_y) \cdot x - (B_x - A_x) \cdot y + B_x \cdot A_y - B_y \cdot A_x\,|}{\sqrt{(B_y - A_y)^2 + (B_x - A_x)^2}}. \tag{1.3}$$

We also need to know on which side of the line the robot is. We note s this information, with $s \in \{-1, 1\}$. So :

$$s = sign((x - A_x)\ (B_y - A_y) - (y - A_y)\ (B_x - A_x)). \tag{1.4}$$

Finally, we can compute ϕ, the angle the robot should have in order to follow the line :

$$\phi = \theta + \arctan(\frac{s \cdot d}{k}), \tag{1.5}$$

where k is a coefficient which defines the strength of the curve : it is the distance from the line where the angle with the line is 45^o.

An example of vector field is presented on Figure 1.3.

Despite the number of attempts before having a satisfying behavior, at the end the boat well followed the lines. Therefore it could participate at *La Cordelière* research, by making lines in the *Rade de Brest* with appropriate sensors, to help the *André Malraux* (see Figure 1.4) in the places it can't go, because of the shallows.

An example of the user interface while Boatbot is following lines is presented on Figure 1.5. We can see the track of the boat in comparison with the desired lines. So we notice that the robot is always close to the lines, and has a straight direction. However, it is often lightly displaced and not exactly on the line. This is mostly due to the currents, and that is why we will try to estimate them in the next chapter in order to take them into account in the control.

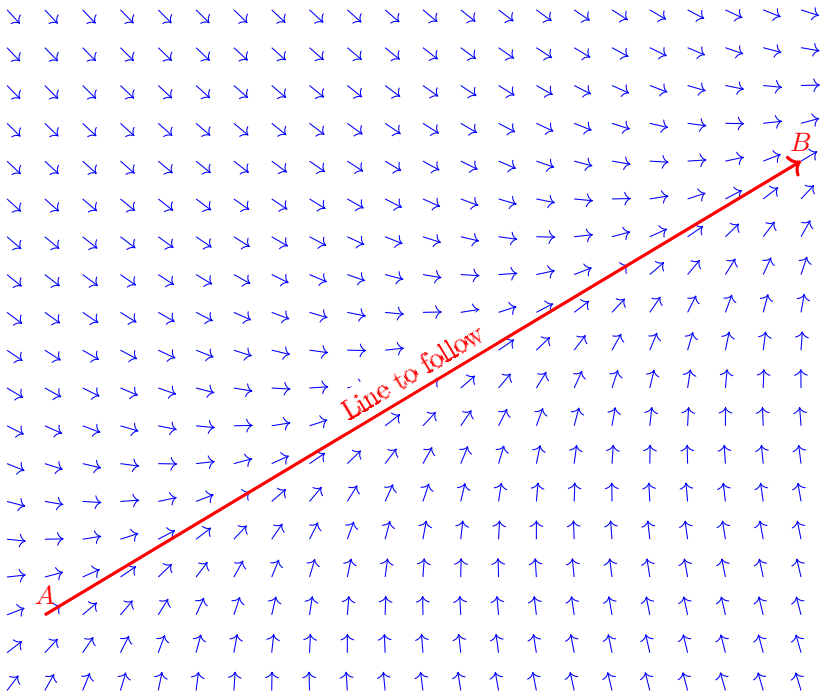


Figure 1.3: Example of vector field to follow the line ($A \rightarrow B$).

Figure 1.4: Picture of *André Malraux* researching *La Cordelière*. *André Malraux* is a boat of the DRASSM.

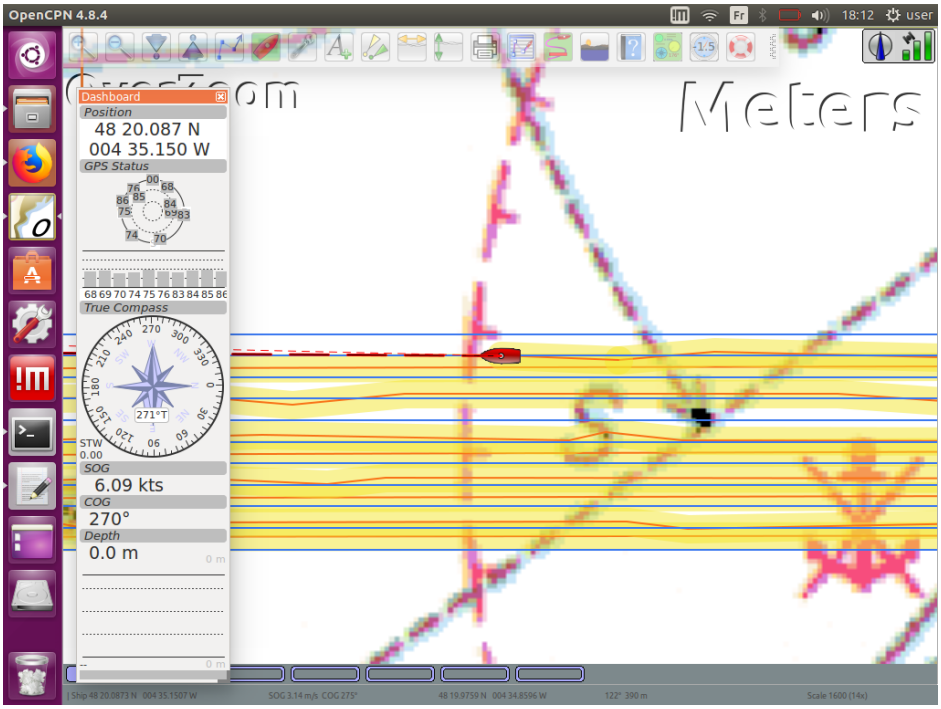


Figure 1.5: Screenshot of the user interface for Boatbot, after having made some lines. The lines to follow are in blue, and the track of the boat is in orange, highlighted in yellow.

Chapter 2

Estimation of the current with a Kalman filter

The current is a perturbation which is difficult to evaluate and must be taken into account to have a proper control. If we can estimate this perturbation, the control will be easier and better. We can consider that the current is constant in a short period, and therefore think using an integral term in the controller to compensate the static error. But this error is static only if the boat always follow the same heading, and most of the time, that is not the case.

An interesting way to estimate this current, whatever the speed or the heading of the boat, is to use a Kalman filter. Indeed, the Kalman filter is used to estimate unknown variables (here the currents) by using several measures containing statistical noise (here the speed and the course over ground and the heading)[7].

2.1 Description of the model

The boat is described by the following state equations :

$$\begin{cases} \dot{x_1} &= p_1 \cos(x_3) + p_2 \\ \dot{x_2} &= p_1 \sin(x_3) + p_3 \\ \dot{x_3} &= u \end{cases} \tag{2.1}$$

where (x_1, x_2) are the coordinates of the robot (its center), x_3 its heading, p_1 the speed of the boat with respect to the water, and (p_2, p_3) the unknown currents.

Here we want to estimate the vector $\boldsymbol{p} = (p_1, p_2, p_3)^T$ which is unknown. We consider that the boat speed is constant, and that the currents change very slowly, so they are constant in our scale. So, we have :

$$\dot{\boldsymbol{p}} = \boldsymbol{0} + \boldsymbol{\alpha}(t), \tag{2.2}$$

and we measure the speed and the course over ground, which include the robot speed added to the currents :

$$\boldsymbol{y} = \begin{pmatrix} p_1 \cos(x_3) + p_2 \\ p_1 \sin(x_3) + p_3 \end{pmatrix} + \boldsymbol{\beta}(t), \tag{2.3}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are random independent Gaussian noises.

So, we can now get a reliable estimation $\hat{\boldsymbol{p}}$ of $\boldsymbol{p}$ using a Kalman filter.

2.2 Kalman filter

Let us write the state equations with matrix, and discretized :

$$\left\{ \begin{array}{rcl} \boldsymbol{p}_{k+1} & = & \boldsymbol{A}_k \boldsymbol{p}_k + \boldsymbol{\alpha}_k \\ \boldsymbol{y}_k & = & \boldsymbol{C}_k \boldsymbol{p}_k + \boldsymbol{\beta}_k \end{array} \right. , \tag{2.4}$$

with : $\boldsymbol{A}_k = \boldsymbol{I}$ (the Identity), and $\boldsymbol{C}_k \overset{2.3}{=} \begin{pmatrix} \cos(x_3) & 1 & 0 \\ \sin(x_3) & 0 & 1 \end{pmatrix}$.

So, now we can write the complete Kalman filter and compute the estimation $\hat{\boldsymbol{p}}$ with the two phases : *correction* and *prediction* [7].

Finally, we have :

$$\begin{array}{rcll} \hat{\boldsymbol{p}}_{k+1|k} & = & \boldsymbol{A}_k \hat{\boldsymbol{p}}_{k|k} & \text{(predicted estimation)} \\ \boldsymbol{\Gamma}_{k+1|k} & = & \boldsymbol{A}_k \cdot \boldsymbol{\Gamma}_{k|k} \cdot \boldsymbol{A}_k^T + \boldsymbol{\Gamma}_{\boldsymbol{\alpha}_k} & \text{(predicted covariance)} \\ \hat{\boldsymbol{p}}_{k|k} & = & \hat{\boldsymbol{p}}_{k|k-1} + \boldsymbol{K}_k \cdot \tilde{\boldsymbol{y}}_k & \text{(corrected estimation)} \\ \boldsymbol{\Gamma}_{k|k} & = & (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{C}_k) \cdot \boldsymbol{\Gamma}_{k|k-1} & \text{(corrected covariance)} \\ \tilde{\boldsymbol{y}}_k & = & \boldsymbol{y}_k - \boldsymbol{C}_k \hat{\boldsymbol{p}}_{k|k-1} & \text{(innovation)} \\ \boldsymbol{S}_k & = & \boldsymbol{C}_k \cdot \boldsymbol{\Gamma}_{k|k-1} \cdot \boldsymbol{C}_k^T + \boldsymbol{\Gamma}_{\boldsymbol{\beta}_k} & \text{(covariance of the innovation)} \\ \boldsymbol{K}_k & = & \boldsymbol{\Gamma}_{k|k-1} \boldsymbol{C}_k^T \boldsymbol{S}_k^{-1} & \text{(Kalman gain)} \end{array} .$$

2.3 Control

Now, we want to implement a control of the the robot based on a classic feedback linearization. Let us take as an output

$$y = x_3 + \arctan(x_2). \tag{2.5}$$

We want our controller to make the output y converges to 0, so the robot will perform a line following (y is seen as the error).

By differentiating 2.5, we get :

$$\begin{array}{rcll} \dot{y} & = & \dot{x_3} + \dfrac{\dot{x_2}}{1 + x_2^2} & (2.6) \\ & \overset{2.1}{=} & u + \dfrac{p_1 \sin(x_3) + p_3}{1 + x_2^2}. & (2.7) \end{array}$$

So, as the differential equation 2.7 is of order 1, we choose a first order equation for the error y :

$$\dot{y} + y = 0. \tag{2.8}$$

The command u should have this error equation satisfied, hence :

$$\begin{array}{rcll} u & = & -y - \dfrac{p_1 \sin(x_3) + p_3}{1 + x_2^2} & (2.9) \\ & \overset{2.5}{=} & -x_3 - \arctan(x_2) - \dfrac{p_1 \sin(x_3) + p_3}{1 + x_2^2}. & (2.10) \end{array}$$

This control will make the robot attracted by the line $x_2 = 0$. We can remark that there is not any singularity.

As we do not know the currents and the speed of the robot, we use the estimation given by the Kalman filter presented in Section 2.1 :

$$u = -x_3 - \arctan(x_2) - \frac{\hat{p_1}\sin(x_3) + \hat{p_3}}{1 + x_2^2}. \tag{2.11}$$

2.4 Simulation and results

The controller presented in Section 2.3 has been implemented into a simulation in two dimensions, with Python. The code is available in Appendix B. The aim of this simulation is to prove that the currents are well estimated by the Kalman filter, and that the control works for a line following.

So the results are convincing : on Figure 2.1, we can see that the direction of the current is already quite good, the lack of accuracy is mainly on the strength of this current. And quickly, it is very well estimated, with high accuracy, since the Figure 2.2 shows the same simulation a few seconds later. We also notice that the robot is on the line, and moves like a crab, since it takes the currents into consideration.

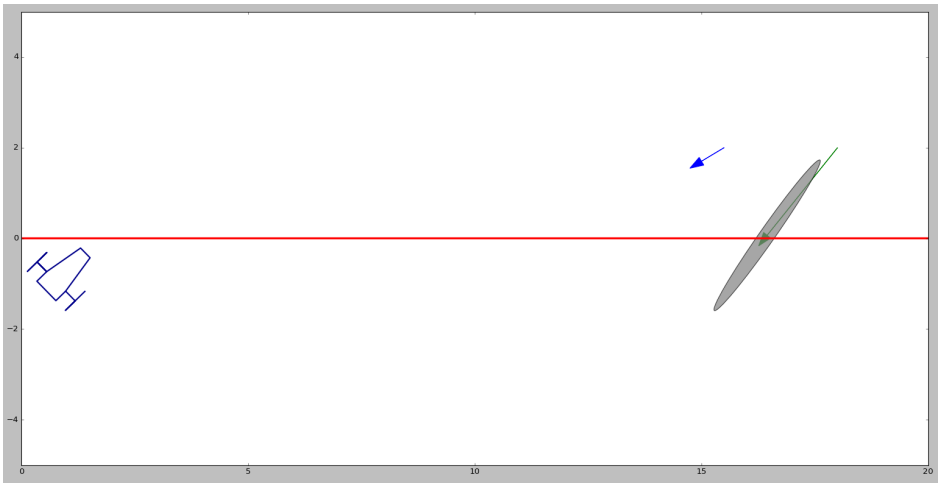


Figure 2.1: Screenshot of the simulation at the beginning. The blue array represents the real current whereas the green one is the estimated one. The gray ellipse represents the incertitude of the estimation, it is a representation of the covariance matrix given by the Kalman filter.

This simulation shows good results, and let think this method works well. However, it has not been implemented and tested on a real robot yet, so we can not conclude for now. Indeed, in the simulation, the model used is relatively simple, it is linear and Gaussian, and it is exactly the same model for the simulation and in the Kalman filter. Therefore, we stacked all the odds in our favor to have the expected results, but even if the results are encouraging, we should wait for real tests before drawing a reliable conclusion.

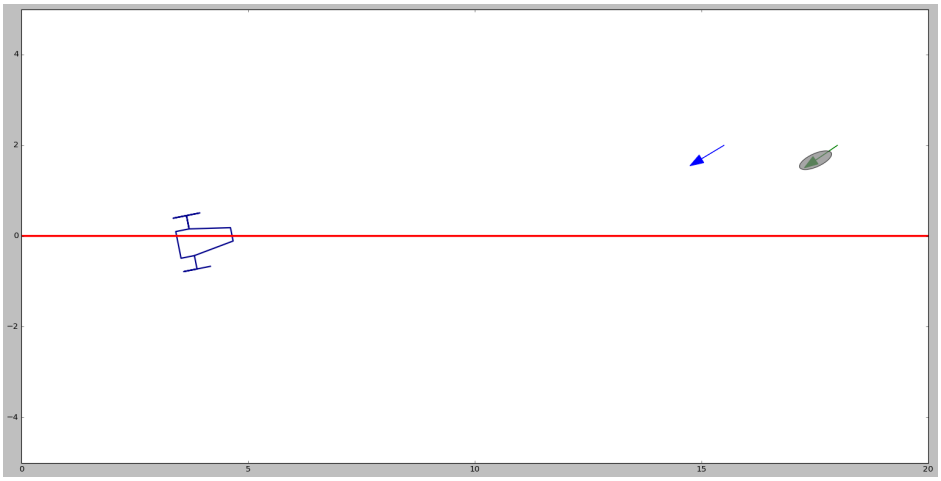


Figure 2.2: Screenshot of the simulation once the current is well estimated.

2.5 Tight slalom control for sailboat robots

The control presented in Section 2.3 is based on an associated vector field, as illustrated on Figure 2.3. It works well for a line following, and it is precise. This approach can be generalized to any vector field.

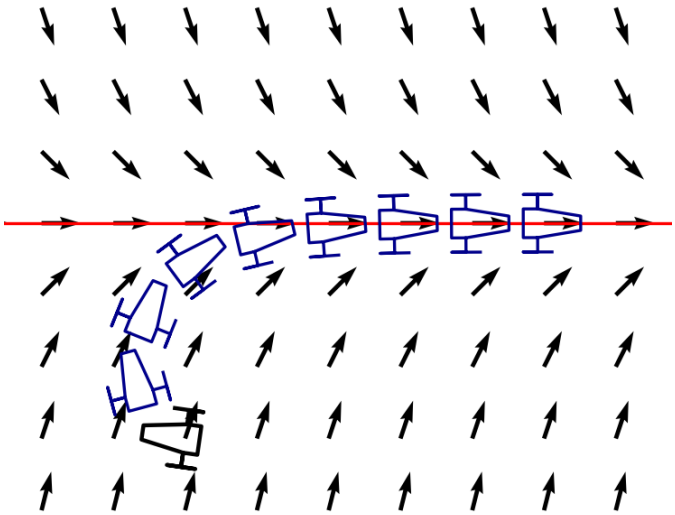


Figure 2.3: Line following with its associated vector field.

What is important in this approach is that we do not control the speed, we only regulate the heading and show that it is sufficient for following a precise path accurately. The difference between this kind of control and a classic linear controller is that we just want to be collinear to the vector field instead of being equal. Here, the controller anticipates the fact that the required trajectory have to take into account the curvature of the vector field.

This approach has been developed in order to control a sailboat, where the speed is generally hard to control. This was in the context of the participation in WRSC[1] and IRSC[2]. We wrote an article to explain how does it work for the IRSC [9]. The article is joined to this report, in Appendix C.

So we simulated a test-case where a tight slalom is performed by a sailboat robot (see Figure 2.4), showing that by anticipating the required trajectory, we succeed in following exactly a vector field, and, in this example, succeed at a tight slalom without missing any gate.

[1] WRSC : World Robot Sailing Championship, a competition for non-motorized autonomous boats, up to 4m long, using only wind as propulsion.

[2] IRSC : International Robotic Sailing Conference.

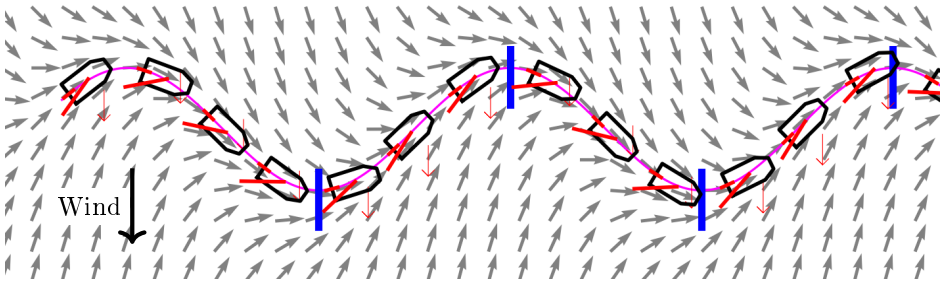


Figure 2.4: Simulation of a sailboat performing a tight slalom.

Chapter 3

Bathymetric localization of an AUV

Even when we have a relatively precise idea of where is a researched wreck, and it is not easy to have, a huge amount of work is still much to do. Indeed, the research zone is very often spread, and most of the used sensors require to move slowly. It represents days of navigation on the sea, and sometimes we are not sure to find the wreck. Moreover, if the researched boat sank a very long time ago, it might be completely covered by sediments.

All of these elements raise two issues. The first one depends on the place the research takes place, but if it is in a high frequented place (with a lot of maritime traffic), as in the *Rade de Brest*, it can be difficult and disturbing to lock down the place during a long time. The second one is about sensors. If the wreck is buried under several meters, the only sensor I know which might detect the boat is a magnetometer. However, the magnetometer should be towed very close to the seabed, in order to be the closest of the wreck as possible. But when the sensor is towed from a boat, it is very difficult to regulate its depth and to know where it is accurately.

In order to tackle these issues, a good solution is to use underwater vehicles, on condition that it can well localize itself. Indeed, it is not a matter of having a boat connected to the vehicle or of needing the vehicle to resurface regularly. It must be completely autonomous.

3.1 Use of the bathymetric information of the place

Today, the problem of localization is often quickly solved by using a GPS. It is cheap and effective. However, GPS uses radio waves and so the signal is stopped by walls and water. But in such places like inside buildings or in the water, we want to see robots work. And as a good control depends on a good localization, the localization issue is still topical in the research field. For now, under the water, the common solution consists in getting a GPS fix, diving, and using dead reckoning to estimate the position. But after a while, the robot must resurface to have a new GPS fix, otherwise the cumulated errors will be too important and the robot will be quickly lost.

The idea exposed here is about using the bathymetry, that is the measurement of the depths of the seas, in order to find where the robot can be. At the beginning, the robot does not know where it is, but it knows the bathymetric map of the surrounding area. It moves, while it measures the depth regularly, with a sonar for instance, and it tries to find where are the possible places in the map that correspond to its measurements, and by taking into account the eventual measurement errors.

So we suppose we already have the bathymetric map of the interesting region. Actually, it is something relatively easy to find today, the hardest is to have a good resolution. Otherwise, we

can imagine an AUV doing bathymetric SLAM[1], like in [3]. However, it supposes that the AUV has at least a multi-beam echo sounder, which is a big and expensive sensor.

3.2 Use of a binary tree to store the bathymetric map

Now, the point is that we have a bathymetric map of the whole area to use. As the map is high defined – we hope so – it represents a big file with a huge amount of data. So the idea is to place data in a tree so that we can load only a part of the map in each iteration, and earn a lot of time and resources.

The tree is built so that each node of the tree contains the interval of depths included in the region the node represents. It means that the root, which contains the whole map, will store the interval that include every existing depths in the map. As the tree is binary, the root will have two sons : the left part of the map, and the right one. So, for instance, if our robot measures a depth of 12 meters, and the left son contains the interval $[8, 15]$, and the right one the interval $[14, 22]$, then only the left part of the map will be loaded, because we know that we can not be in the right one. And this process is continued in each interesting branch of the tree, so that at the end the calculations are made only on some parts of the map.

An example of a localization after only one iteration is provided on Figure 3.1. We can see on the left a representation of the bathymetric map, and on the right the result after simulating one measure on this map. To be precise, the measure chosen here is 17 meters of depth, with 0.3 meter of estimated error.

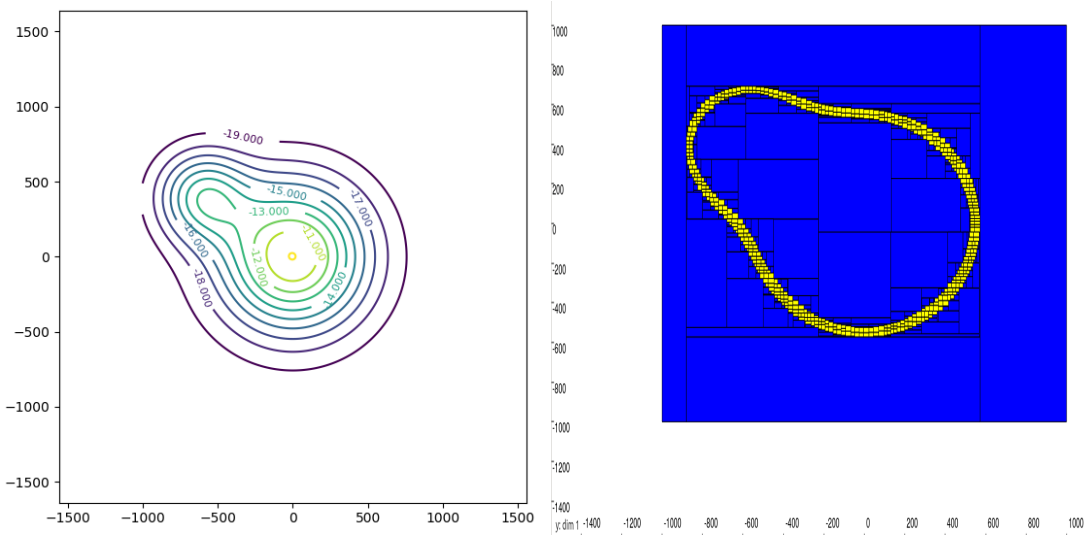


Figure 3.1: Bathymetric localization.

[1]SLAM : Simultaneous Localization And Mapping.

3.3 Interval analysis and contractors

So, in our context, we are working with intervals : the binary tree presented in Section 3.2 stores depths in interval. The aim is to use these intervals and make some calculations on them to facilitate the localization. Actually, the robot will know its position only by computing a *box* which includes its real position. A *box* is an interval vector. Here, as the robot is localized in two dimensions, the researched box is the Cartesian product of two intervals : $[x]$ and $[y]$.

In order to improve the localization, we will see that we can contract the boxes to reduce their size [8]. A *contractor* is an operator associated with a constraint. This contractor is applied on a box, which one will be reduced such that the *contracted box* is included in the initial box, and the constraint is still satisfied. A contraction is illustrated on Figure 3.2.

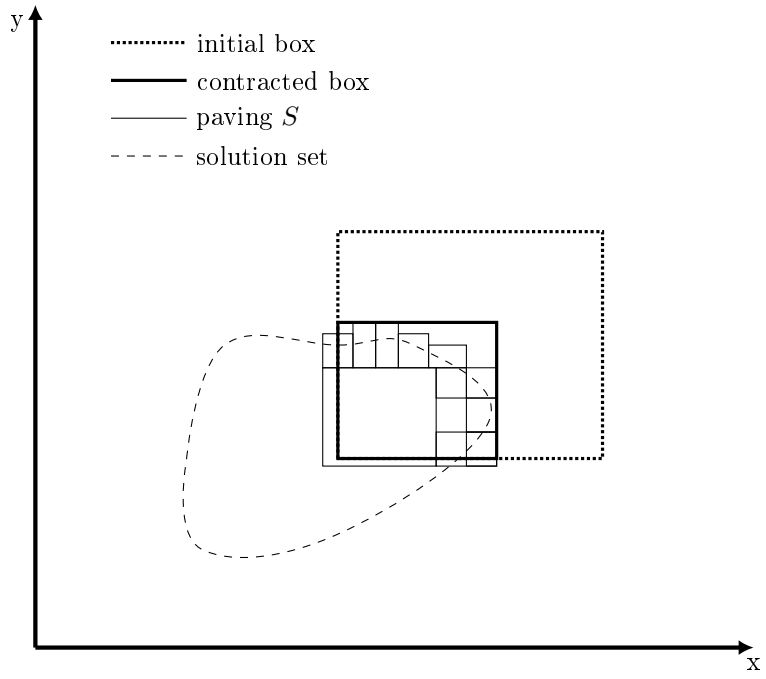


Figure 3.2: Contraction of a box.

3.4 Localization

Finally, the algorithm for the localization is quite simple. The first step is explained in Section 3.2 : the robot measures the depth, and find a union of boxes in the map where it can be. Then, the robot will move a little, and a new iteration begins. Here we have two new informations : a new measure of depth, and a moving which can be computed like in dead reckoning, by using IMU information and estimating the move. This last estimation is marred by the measurement errors, but is used to transform the box we have at the last iteration into a new box, larger because of the errors, which is an approximation of our new position. Then, this new box will be contracted using the new measure of depth and the bathymetric map as constraint. Finally, we reiterate this continuously, and the box will be smaller and smaller, therefore the localization will be better.

Obviously, this method works only if the seabed is not flat, and if the relief is irregular enough.

Conclusion

To conclude, my internship was very varied. I worked on a real system, Boatbot, and had an overview of all the difficulties working on a real robot implies. Even for a simple control which works very quickly in simulation, the step to the real robot is really big.

I used two approaches for estimating an unknown variable : a Kalman filter for the currents, and interval analysis for a bathymetric localization. Both of them are interesting, and enable different ways to tackle a problem.

Finally, I took part in the writing of an article and led some simulations in order to present a new approach for controlling sailboat robots.

Now, there are several foreseeable areas of work. The algorithm for the estimation of the currents with the Kalman filter has not been implemented on a real robot yet. That would be interesting, or at least to test it in simulation but with a realer model.

Concerning the bathymetric localization, there is still much work to do, and I will continue to develop this approach in particular during a thesis.

Bibliography

[1] Wrsc &irsc 2018 - southampton. https://www.roboticsailing.org/2018/, 2018.

[2] La Région Bretagne. À la recherche de *la Cordelière*. https://www.lacordeliere.bzh/, 2018.

[3] Benoît Desrochers. Slam in unstructured environments ; a set-membership approach. Master's thesis, ENSTA Bretagne (Lab-STICC, FR), 2018.

[4] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011.

[5] Open Source Robotics Foundation. Ros wiki. http://wiki.ros.org/, 2018.

[6] Auguste Jal. *Marie-La-Cordelière*. 1845.

[7] Luc Jaulin. *Mobile Robotics*. 2015.

[8] Olivier Didrit Luc Jaulin, Michel Kieffer and Éric Walter. *Applied Interval Analysis*. 2001.

[9] Luc Jaulin Maël Le Gallic, Joris Tillet and Fabrice Le Bars. Tight slalom control for sailboat robots. *IRSC 2018*, 2018.

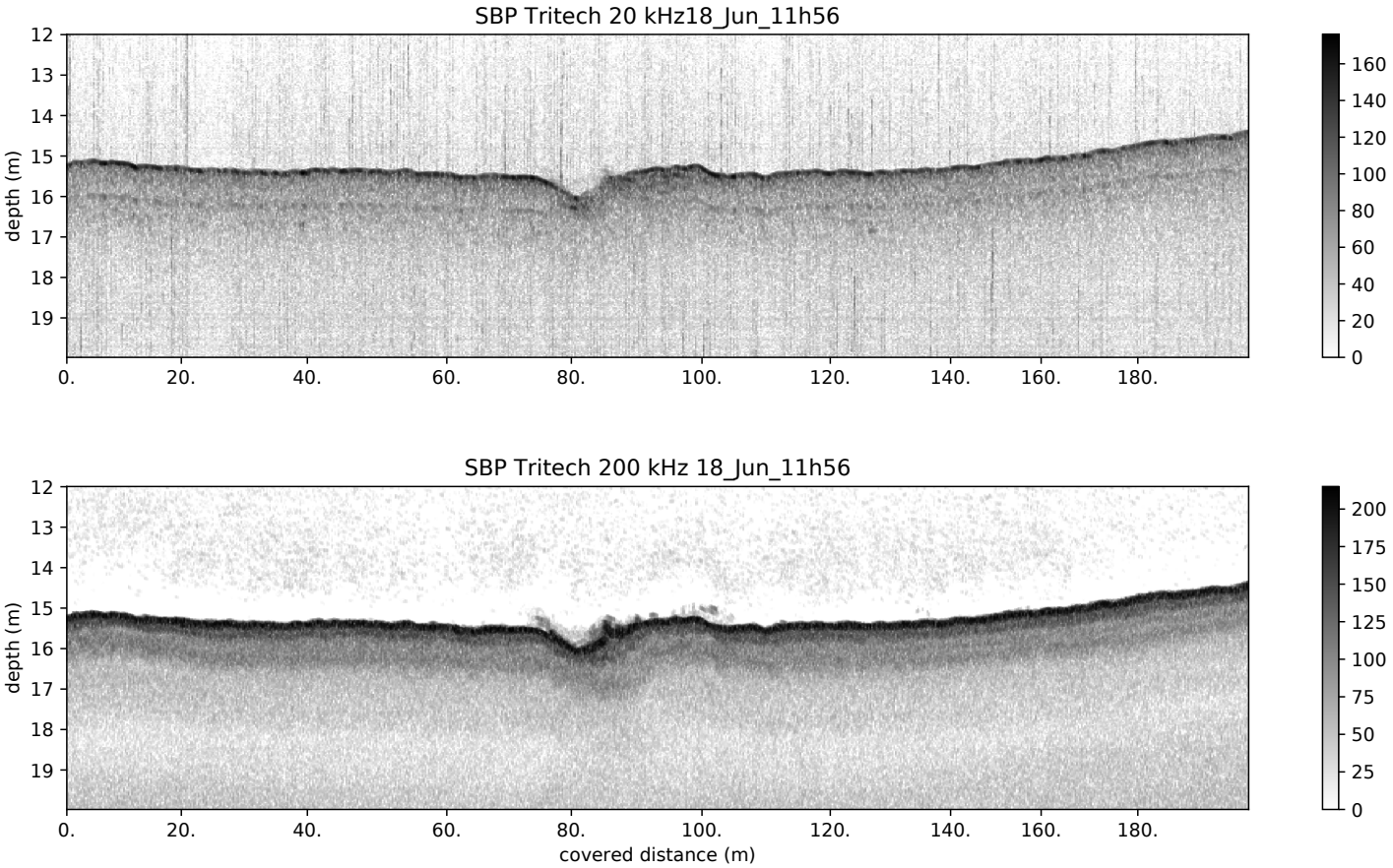
[10] Eric Rieth et Christophe Cérino Michel l'Hour. *Archéologie sous-marine : pratiques, patrimoine, médiation*. Presses Universitaires de Renne, 2013.

Appendix A

Boatbot : examples of data from the SBP and the magnetometer

A.1 Example of data from the SBP (with detection of a wreck)

The SBP, or sediments sounder, give two graphs, corresponding to two different frequencies. Indeed, the sensor receives a high frequencies signal, represented here on the bottom graph. It gives information about the seabed, and do not go through the ground. So we see the depth and whether a wreck sticks out. But what is particular with this sensor is the low frequency signal, which is represented in the top graph. As it is low frequencies, it can go through the different sediment layers, and evidence something buried under some meters of sediments. In the graph, we can see black points under the seabed, so it is certainly due to tough parts of a buried wreck.


SBP Tritech 20 kHz18_Jun_11h56
depth (m)
12
13
14
15
16
17
18
19
0.
20.
40.
60.
80.
100.
120.
140.
160.
180.
160
140
120
100
80
60
40
20
0
SBP Tritech 200 kHz 18_Jun_11h56
depth (m)
12
13
14
15
16
17
18
19
0.
20.
40.
60.
80.
100.
120.
140.
160.
180.
covered distance (m)
200
175
150
125
100
75
50
25
0

A.2 Example of data from the magnetometer (with detection of a wreck)

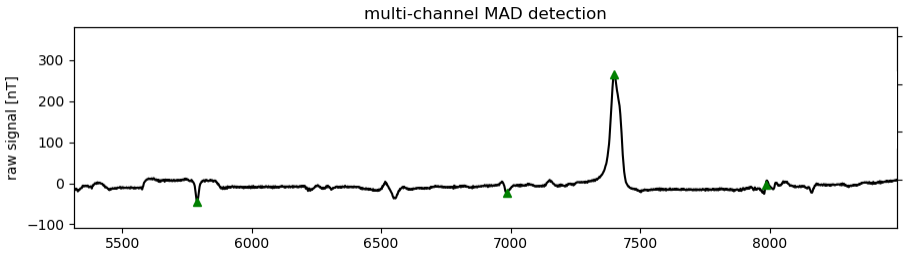


Figure A.1: Example of a detection with the magnetometer. This is the amplitude of the magnetic field according to the time. The green triangles corresponds to a detection by the algorithm. Here, the big peak is caused by a known wreck.

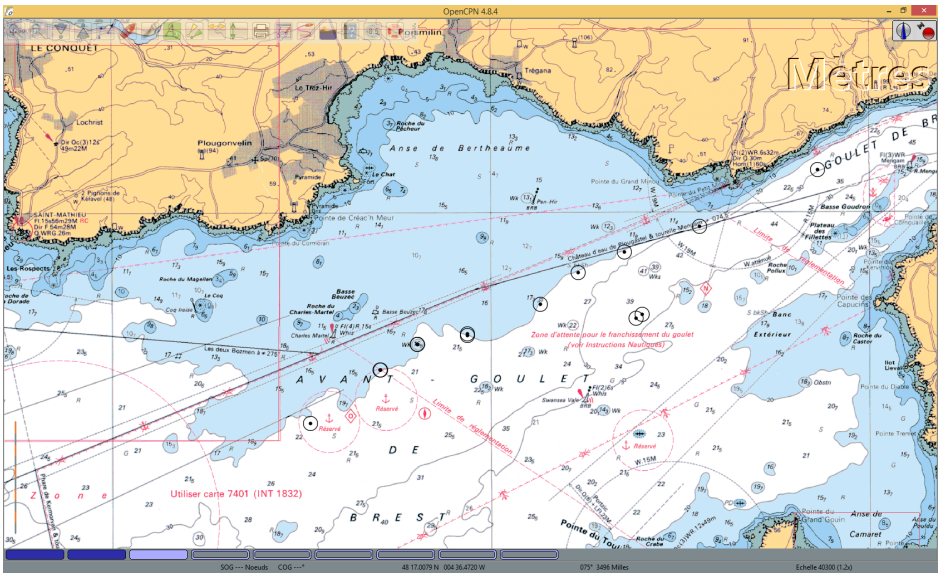


Figure A.2: Map of targets. Once the positive detections of the magnetometer have been linked with GPS fixes, a map like this one is exported to localize the targets where there were the detections.

Appendix B

Python code of the simulation for estimating the currents

```
"""
This code displays a simulation of a robot being controlled to follow a line.
The currents modify the trajectory of the robot, so a Kalman filter is used to
estimate these currents, and therefore the estimation is used in the control.
"""

from roblib import * # Local library containing usefull functions for plotting

p1 = 1.0 # Speed of the robot
p2 = -0.2 # Speed of the currents with respect to x
p3 = 0.3 # Speed of the currents with respect to y
gamma = 100*eye(3, 3) # Initial covariance matrix of estimation of vector p
gamma_a = 0.000001 # Incertitude on alpha
gamma_b = 0.2 # Incertitude on beta

def f(x, u):
    x = x.flatten()
    theta = x[2]
    xdot = np.array([[p1*cos(theta) + p2], [p1*sin(theta) + p3], [u]])
    return xdot


def draw_target(xmin, xmax):
    s = arange(xmin, xmax, 0.01)
    w = array([s, 0*sin(s)])
    plot2D(w, "red", 3)
    return

def kalman_predict(xup,Gup,u,gamma_alpha,A):
    gamma_1 = A @ Gup @ A.T + gamma_alpha
    x1 = A @ xup + u
    return(x1,gamma_1)
```

```
def kalman_correc(x0,gamma_0,y,gamma_beta,C):
    S = C @ gamma_0 @ C.T + gamma_beta
    K = gamma_0 @ C.T @ inv(S)
    ytilde = y - C @ x0
    Gup = (eye(len(x0))-K @ C) @ gamma_0
    xup = x0 + K @ ytilde
    return(xup,Gup)

def kalman(x0,gamma_0,u,y,gamma_alpha,gamma_beta,A,C):
    xup,Gup = kalman_correc(x0,gamma_0,y,gamma_beta,C)
    x1,gamma_1=kalman_predict(xup,Gup,u,gamma_alpha,A)
    return(x1,gamma_1)

def control_kalman(x, phat, u):
        global gamma
        x1, x2, x3 = x.flatten()
        C = array([[cos(x3), 1, 0], [sin(x3), 0, 1]])
        y = C @ array([[p1], [p2], [p3]]) + gamma_b*randn(2,1) # measure
        phat, gamma = kalman(phat, gamma, 0*u, y, gamma_a*eye(3, 3),
                                                  gamma_b*eye(2, 2), eye(3,3), C)
        u = -x3 - arctan(x2) - (phat[0][0]*sin(x3)+phat[2][0])/(1+(x2)**2)
        return phat, u

# Initial state
x = array([[0], [-3], [1]])
u = 0

phat = array([[0], [0], [0]])
dt = 0.05

xmin, xmax, ymin, ymax = 0, 20, -5, 5
ax = init_figure(xmin, xmax, ymin, ymax)

# Boucle of simulation : the Euler scheme is used to simulate the behaviour of
# the robot.
for t in arange(0, 12, dt):
        clear(ax)
        draw_tank(x, 'darkblue', 0.3) # Draw a robot at the right position
        draw_target(-xmin, xmax)
        arrow(15.5, 2, p2, p3, head_width=0.2, color='blue') # Current
        arrow(18, 2, phat[1][0], phat[2][0],
                  head_width=0.2, color='green') # Estimated current
        draw_ellipse(array([[18+phat[1][0]],[2+phat[2][0]]]), gamma[1:, 1:], 0.9,
                                ax, 'gray') # Covariance matrix
        phat, u = control_kalman(x, phat, u)
        x = x+dt*f(x, u) # Euler
pause(1)
```

Appendix C

The article : Tight slalom control for sailboat robots

Tight slalom control for sailboat robots

Mael Le Gallic
ENSTA Bretagne
mael.le_gallic@ensta-bretagne.org

Joris Tillet
ENSTA Bretagne
joris.tillet@ensta-bretagne.org

Fabrice Le Bars
Lab-STICC
ENSTA Bretagne
fabrice.le_bars@ensta-bretagne.fr

Luc Jaulin
Lab-STICC
ENSTA Bretagne
lucjaulin@gmail.com

Abstract

Existing controllers for sailboat robots are usually developed for speed performances and for long straight lines. In this context, the accuracy is not the main concern. In this paper, we consider the tight slalom problem which requires accuracy. We propose a feedback-linearization based method combined with a vector field approach to control the sailboat. Some simulations show that the robot is able to perform the slalom without missing any gate.

1 Introduction

We consider a mobile robot described by the state equations [5]

$$\left\{ \begin{array}{rcl} \dot{\mathbf{x}} & = & \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{p} & = & \mathbf{g}(\mathbf{x}) \end{array} \right. \tag{1}$$

with an input vector $\mathbf{u} = (u_1, \ldots, u_m)$, a state vector $\mathbf{x} = (x_1, \ldots, x_n)$ and a pose vector $\mathbf{p} = (p_1, \ldots, p_{m+1})$ with $n \geq m + 1$. The goal of this paper is to show we can follow a chosen vector field in the $\mathbf{p}$ space [8][12][13], using a feedback-linearization based method. It means that we can control $m + 1$ state variables and not only m of them, as given by the theory [4]. This is due to the fact that we perform a path following instead of a trajectory tracking where the time is involved. In practice, the vector $\mathbf{p}$ corresponds to the position of the center of the robot and may be of dimension 2 (if $m = 1$) or 3 (if $m = 2$). This is consistent with the fact that we need one actuator to control the direction of a 2D vehicle such as a car or a boat and two actuators for a 3D vehicle such as a plane.

The approach we propose is to find a controller so that the vector $\dot{\mathbf{p}}$ be collinear (instead of equal) to the required field. This is illustrated in this paper in the case where the mobile robot is a sailboat [10][2][9]. The input $\mathbf{u}$ is scalar (*i.e.*, $m = 1$) and corresponds to the rudder. Moreover, we will show that this approach is particularly adapted to sailboats where the speed is hardly controllable [11][14].

2 Method

In order to facilitate the understanding of our approach, we will deal with a Dubins car, which is much simpler than a sailboat. The extension to other type of mobile robots is straightforward.

In: S. M. Schillai, N. Townsend (eds.): Proceedings of the International Robotic Sailing Conference 2018, Southampton, United Kingdom, 31-08-2018

2.1 Line following for a Dubins car

To introduce our approach, we consider a robot (here a Dubins car) moving on a plane and described by the following state equations:

$$\left\{\begin{array}{rcl}\dot{x}_1 &=& \cos x_3\\ \dot{x}_2 &=& \sin x_3\\ \dot{x}_3 &=& u\end{array}\right. \tag{2}$$

where x_3 is the heading of the robot and $\mathbf{p} = (x_1, x_2)$ are the coordinates of its center. The state vector is given by $\mathbf{x} = (x_1, x_2, x_3)$.

Let us choose as the control output the variable

$$y = x_3 + \text{atan}(x_2). \tag{3}$$

and let us find a classical feedback linearization based controller [6] such that the output y (which can be interpreted as an error) converges to 0. In such a case, we will have

$x_3 + \text{atan}(x_2) = 0$

and the robot will perform a line following. Differentiating (3) we have

$$\dot{y} = \dot{x}_3 + \frac{\dot{x}_2}{1+x_2^2} = u + \frac{\sin x_3}{1+x_2^2}. \tag{4}$$

Since u occurs in (4), the relative degree of the system is 1. We may thus choose a first order equation for the error y, such as

$$\dot{y} + y = 0, \tag{5}$$

We then choose u to have this error equation satisfied. From (4) and (5), we get:

$$\begin{array}{rcl} u &=& -y - \frac{\sin x_3}{1+x_2^2}\\ &=& -x_3 - \text{atan}(x_2) - \frac{\sin x_3}{1+x_2^2}\end{array} \tag{6}$$

Note that we do not have any singularity. As illustrated by the simulation depicted on Figure 1, the associated vector field makes the car attracted by the line $x_2 = 0$.

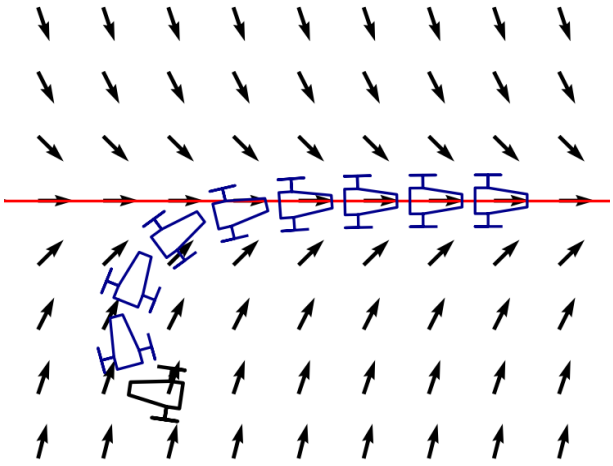


Figure 1: Precise line following

Remark. For more robustness with respect to small uncertainties, a sliding mode effect could be added. It suffices to take for required error

$$y = x_3 + \text{atan}\left(x_2 + \alpha \cdot \text{sign}\left(x_2\right)\right),$$

where α is a small positive coefficient, *e.g.*, $\alpha = 0.1$. In such a case, the robot will go to the line in a finite time (and not asymptotically, as previously). Moreover, it will remains exactly on the line even if some small uncertainties occur.

2.2 Generalization

We want our robot to follow the field $\boldsymbol{\psi}(\mathbf{p})$, more precisely, we want that $\boldsymbol{\psi}(\mathbf{p})$ and $\dot{\mathbf{p}}$ point toward the same direction. This condition can be translated into the form $\boldsymbol{\varphi}\left(\boldsymbol{\psi}(\mathbf{p}), \dot{\mathbf{p}}\right) = 0$, where $\boldsymbol{\varphi}$ is a *collinearity function* which satisfies

$$\boldsymbol{\varphi}\left(\mathbf{r}, \mathbf{s}\right) = \mathbf{0} \Leftrightarrow \exists \lambda > 0,\ \lambda \mathbf{r} = \mathbf{s}. \tag{7}$$

Typically, this function corresponds to one angle (the heading) if $m = 1$ and two angles (heading, elevation) for $m = 2$. Note that the function $\boldsymbol{\varphi}$ cannot be expressed with a determinant since $\mathbf{r}, \mathbf{s}$ should not point toward opposite directions. We define the output

$$\mathbf{y} = \boldsymbol{\varphi}\left(\boldsymbol{\psi}(\mathbf{p}), \dot{\mathbf{p}}\right) = \boldsymbol{\varphi}\left(\boldsymbol{\psi}(\mathbf{g}\left(\mathbf{x}\right)), \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) \cdot \mathbf{f}\left(\mathbf{x}, \mathbf{u}\right)\right). \tag{8}$$

Since $\mathbf{y} \in \mathbb{R}^m$, we can apply a feedback linearization method and we get $\mathbf{y} \to \mathbf{0}$. This means that the robot will follows the required field. Note that we have no control on the speed, which is not our main concern in this paper.

2D case. Consider for instance the case where $m = 1$. We have

$$\boldsymbol{\psi}(\mathbf{p}) = \begin{pmatrix} \psi_1\left(\mathbf{p}\right) \\ \psi_2\left(\mathbf{p}\right) \end{pmatrix}. \tag{9}$$

We take as an output y, the angle between the actual heading vector $\dot{\mathbf{p}} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) \cdot \mathbf{f}\left(\mathbf{x}, \mathbf{u}\right)$ and the desired heading vector given by $\boldsymbol{\psi}(\mathbf{p})$. Denote by $\theta\left(\mathbf{x}\right)$ the argument of the vector $\dot{\mathbf{p}}$. We have

$$\begin{aligned} y \overset{(8)}{=} &\ \text{angle}\left(\boldsymbol{\psi}(\mathbf{g}\left(\mathbf{x}\right)), \tfrac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) \cdot \mathbf{f}\left(\mathbf{x}, \mathbf{u}\right)\right) \\ = &\ \text{angle}\left(\boldsymbol{\psi}(\mathbf{g}\left(\mathbf{x}\right)), \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}\right) \\ = &\ \text{sawtooth}(\theta - \text{atan2}(\underbrace{\psi_2(\mathbf{g}\left(\mathbf{x}\right))}_{b}, \underbrace{\psi_1(\mathbf{g}\left(\mathbf{x}\right))}_{a})) \end{aligned} \tag{10}$$

The *sawtooth* function is given by:

$$\text{sawtooth}(\widetilde{\theta}) = 2\text{atan}\left(\tan\tfrac{\widetilde{\theta}}{2}\right) = \text{mod}(\widetilde{\theta} + \pi, 2\pi) - \pi \tag{11}$$

As illustrated in Figure 2, the function corresponds to an error in heading. The interest in taking an error $\widetilde{\theta}$ filtered by the *sawtooth* function is to avoid the problem of the $2k\pi$ modulus: we would like a $2k\pi$ to be considered non-zero.

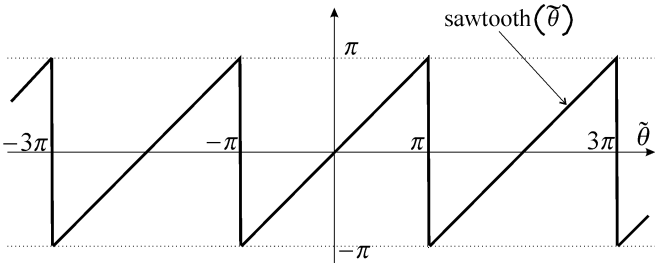


Figure 2: *Sawtooth* function used to avoid the jumps in the heading control

We have

$$\begin{aligned} \dot{y} &= \dot{\theta} - (\underbrace{-\frac{b}{a^2+b^2}}_{\frac{\partial \text{atan2}(b,a)}{\partial a}} \cdot \dot{a} + \underbrace{\frac{a}{a^2+b^2}}_{\frac{\partial \text{atan2}(b,a)}{\partial b}} \cdot \dot{b}) \\ &= u + \tfrac{b \cdot \dot{a} - a \cdot \dot{b}}{a^2+b^2}, \end{aligned} \tag{12}$$

if we assume that the input u corresponds to the desired angular velocity. We propose a feedback linearization based control based on the required equation $\dot{y} = -y$. We have

$$\begin{array}{rcll} u & \overset{(12)}{=} & \dot{y} - \frac{\left(b\cdot\dot{a} - a\cdot\dot{b}\right)}{a^2+b^2} & \\ & = & -y - \frac{\left(b\cdot\dot{a} - a\cdot\dot{b}\right)}{a^2+b^2} & \text{(since } \dot{y} = -y\text{)} \\ & \overset{(10)}{=} & -\left(\text{sawtooth}(\theta - \text{atan2}(b,a)\right) - \frac{\left(b\cdot\dot{a} - a\cdot\dot{b}\right)}{a^2+b^2}. & \end{array} \tag{13}$$

We thus have the guarantee that after some time, the error angle y is 0 and that we follow exactly the vector field.

2.3 Dubins car following the Van der Pol cycle

We would like our Dubins car to follow a path corresponding to the limit cycle of the Van der Pol equation:

$$\boldsymbol{\psi}(\mathbf{p}) = \begin{pmatrix} p_2 \\ -\left(0.01\ p_1^2 - 1\right) p_2 - p_1 \end{pmatrix}. \tag{14}$$

Take $\mathbf{g}(\mathbf{x}) = (x_1, x_2)^{\mathrm{T}}$ which means that we want to build the paths in the (x_1, x_2)-space. We have

$$\boldsymbol{\psi}(\mathbf{g}(\mathbf{x})) = \begin{pmatrix} x_2 \\ -\left(0.01\ x_1^2 - 1\right) x_2 - x_1 \end{pmatrix} \tag{15}$$

and

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \cos x_3 \\ \sin x_3 \\ u \end{pmatrix} \tag{16}$$

Thus

$$\begin{array}{rcc} a & = & x_2 \\ b & = & -(0.01\ x_1^2 - 1)x_2 - x_1 \\ \theta & = & x_3 \end{array} \tag{17}$$

and

$$\begin{array}{rcc} \dot{a} & = & \sin x_3 \\ \dot{b} & = & -\left(0.01 \cdot 2x_1\dot{x}_1\right) x_2 - \left(0.01\ x_1^2 - 1\right)\dot{x}_2 - \dot{x}_1 \\ & = & -0.02 \cdot x_1 x_2 \cos x_3 - \left(0.01\ x_1^2 - 1\right)\sin x_3 - \cos x_3 \end{array} \tag{18}$$

From (13), we get that final controller is

$$\begin{array}{rcl} u & = & -\text{sawtooth}\left(x_3 - \text{atan2}\left(-\left(\frac{x_1^2}{100} - 1\right) x_2 - x_1, x_2\right)\right) \\ & & + \frac{\left(\left(\frac{x_1^2}{100} - 1\right) x_2 + x_1\right)\cdot \sin x_3 + x_2 \cdot \left(\frac{x_1 x_2 \cos x_3}{50} + \left(\frac{x_1^2}{100} - 1\right)\sin x_3 + \cos x_3\right)}{x_2^2 + \left(\left(\frac{x_1^2}{100} - 1\right) x_2 + x_1\right)^2} \end{array} \tag{19}$$

The behavior of the control law is illustrated by Figure 3. The car is very close to the true limit cycle, which is not the case if we consider a classical linear controller. Indeed, the controller anticipates the fact that the required trajectory have to take into account the curvature of the vector field.

3 Application to the slalom problem

We consider the following model which corresponds to a simplified version of the sailboat model given in [7]. The state equations are

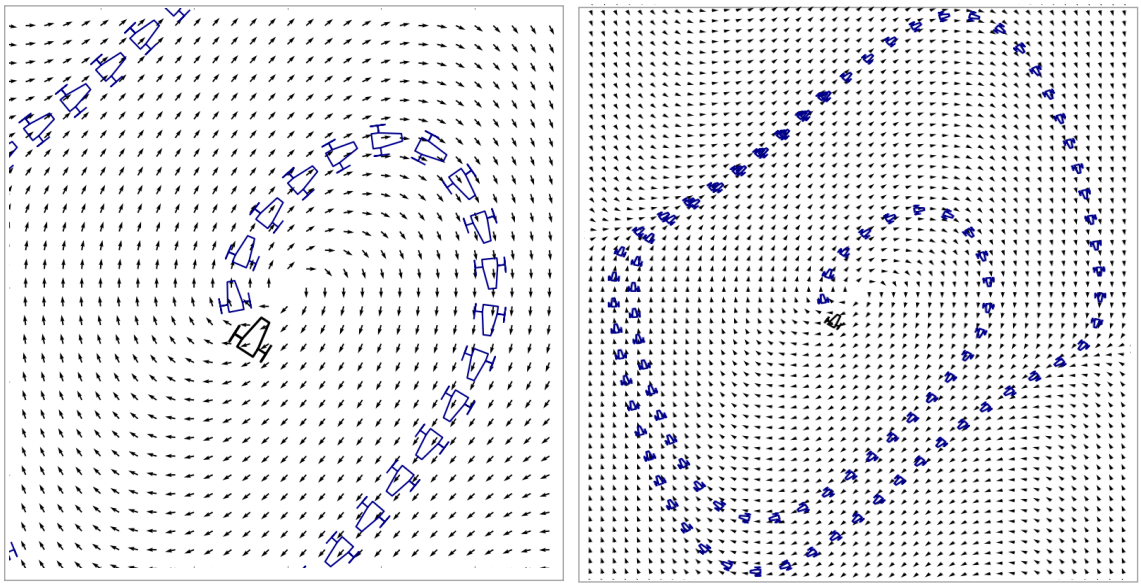


Figure 3: Dubins describing accurately the Van der Pol cycle

$$\left\{\begin{array}{lcl} \dot{x}_1 & = & v\cos\theta \\ \dot{x}_2 & = & v\sin\theta \\ \dot{\theta} & = & -\rho_2 v\sin 2u_1 \\ \dot{v} & = & \rho_3\left\|\mathbf{w}_{\text{ap}}\right\|\sin\left(\delta_s-\psi_{\text{ap}}\right)\sin\delta_s-\rho_1 v^2 \\ \sigma & = & \cos\psi_{\text{ap}}+\cos u_2 \\ \delta_s & = & \left\{\begin{array}{ll} \pi+\psi_{\text{ap}} & \text{if } \sigma\leq 0 \\ -\text{sign}\left(\sin\psi_{\text{ap}}\right)\cdot u_2 & \text{otherwise}\end{array}\right. \\ \mathbf{w}_{\text{ap}} & = & \left(\begin{array}{c} -a\ \sin\left(\theta\right)-v \\ -a\ \cos\left(\theta\right)\end{array}\right) \\ \psi_{\text{ap}} & = & \text{angle }\mathbf{w}_{\text{ap}}\end{array}\right. \tag{20}$$

where $\rho_1=0.003,\rho_2=0.2,\rho_3=3$. In this equation u_1,u_2 correspond to the tuning of the rudder and the sail, respectively. We would like our robot to follow a path which makes a tight slalom through doors that have to be passed. We assume that we have a Cartesian equation for our path. For instance, we consider that the path is described by

$$e\left(\mathbf{p}\right)=10\sin\left(\frac{p_1}{10}\right)-p_2=0 \tag{21}$$

where $e\left(\mathbf{p}\right)$ corresponds to an error. This path corresponds to a path that should be possible for a normal sailboat robot for crosswind conditions. We take a vector field which corresponds to a pole placement strategy. For instance, we want the error satisfies $\dot{e}=-0.1\,e$, so that it will converge to zero in about 10 sec. Thus

$$\underbrace{\cos\left(\frac{p_1}{10}\right)\dot{p}_1-\dot{p}_2}_{\dot{e}(\mathbf{p})}=-\frac{1}{10}\underbrace{\left(10\sin\left(\frac{p_1}{10}\right)-p_2\right)}_{e(\mathbf{p})} \tag{22}$$

We take $\dot{p}_1=1$, to go to the right. As a consequence, we get the following field:

$$\boldsymbol{\psi}(\mathbf{p})=\left(\begin{array}{c}\dot{p}_1\\ \dot{p}_2\end{array}\right)=\left(\begin{array}{c} 1 \\ \cos\left(\frac{p_1}{10}\right)+\frac{1}{10}\left(10\sin\left(\frac{p_1}{10}\right)-p_2\right)\end{array}\right) \tag{23}$$

which is attracted by the curve $p_2=10\sin\left(\frac{p_1}{10}\right)$.

We have

$$\boldsymbol{\psi}(\mathbf{g}\left(\mathbf{x}\right))=\left(\begin{array}{c} 1 \\ \cos\left(\frac{x_1}{10}\right)+\frac{1}{10}\left(10\sin\left(\frac{x_1}{10}\right)-x_2\right)\end{array}\right) \tag{24}$$

and

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} \cos x_3 \\ \sin x_3 \end{pmatrix}. \tag{25}$$

Thus

$$\begin{array}{rcl} a & = & 1 \\ b & = & \cos\left(\frac{x_1}{10}\right) + \sin\left(\frac{x_1}{10}\right) - \frac{1}{10} x_2 \end{array} \tag{26}$$

and

$$\begin{array}{rcl} \dot{a} & = & 0 \\ \dot{b} & = & -\dot{x}_1 \frac{1}{10} \sin\left(\frac{x_1}{10}\right) + \dot{x}_1 \frac{1}{10} \cos\left(\frac{x_1}{10}\right) - \frac{1}{10} \dot{x}_2 \\ & = \frac{1}{10} & \cos x_3 \cdot \left(\cos\left(\frac{x_1}{10}\right) - \sin\left(\frac{x_1}{10}\right)\right) - \frac{1}{10} \sin x_3. \end{array} \tag{27}$$

From (13), we get that the desired angular velocity should be

$$\hat{\omega} = -\left(\text{sawtooth}(\theta - \text{atan2}(b, a)) - \frac{(b \cdot \dot{a} - a \cdot \dot{b})}{a^2 + b^2}\right). \tag{28}$$

Now, since the true angular velocity is $\dot{\theta} = -\rho_2 v \sin 2u_1$, we take

$$u_1 = -\frac{1}{2} \arcsin\left(\tanh\left(\frac{\hat{\omega}}{\rho_2 v}\right)\right). \tag{29}$$

The saturation function *tanh* is needed since the rudder cannot respond to any required $\hat{\omega}$. Indeed, if our controller ask to turn too fast for the boat, $\frac{\hat{\omega}}{\rho_2 v}$ will be more than 1, and the rudder can only do its best. The behavior of our controller is illustrated by Figure 4, where the sailboat has to slalom tightly between doors. We can see that the trajectory follows exactly the sine path (magenta).

The Python source codes associated to the simulation can be found at:

https://www.ensta-bretagne.fr/jaulin/slalompy.zip

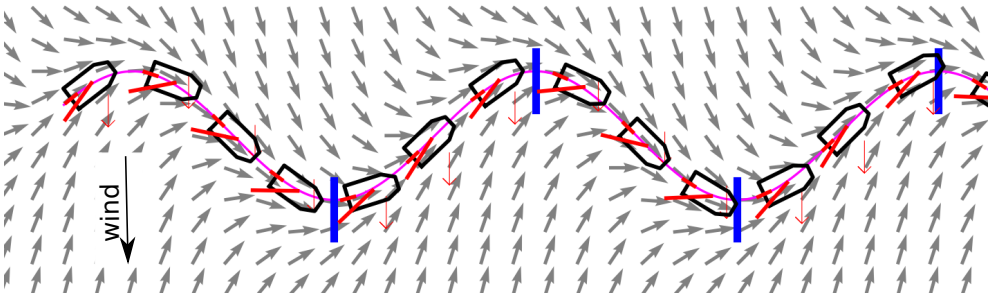


Figure 4: The sailboat robot slaloms through the blue doors

4 Conclusion

In this paper, we have proposed a new controller for sailboat robots which allows to take into account the curvature of the required field in order to anticipate as much as possible the required trajectory. To our knowledge, this is not considered by existing controllers [1] which are devoted to straight lines [3]. It has been shown that the required vector field could be followed exactly. This anticipation is crucial if we want to maneuver quickly and precisely as needed when we want to avoid an obstacle. This has been illustrated on a simulated test-case where a tight slalom is performed by a sailboat robot.

References

[1] F. Le Bars and L. Jaulin. An experimental validation of a robust controller with the VAIMOS autonomous sailboat. In *5th International Robotic Sailing Conference*, pages 74–84, Cardiff, Wales, England, 2012. Springer.

[2] N.A. Cruz and J.C. Alves. Ocean sampling and surveillance using autonomous sailboats. In *International Robotic Sailing Conference*, Austria, 2008.

[3] F. Le Bars F. PLumet, Y. Briere. Les voiliers robotiss. *Les Techniques de l'Ingnieur*, 2018.

[4] A. Isidori. *Nonlinear Control Systems: An Introduction, 3rd Ed.* Springer-Verlag, New-York, 1995.

[5] L. Jaulin. *Automation for Robotics*. ISTE editions, 2015.

[6] L. Jaulin. *Mobile Robotics*. ISTE editions, 2015.

[7] L. Jaulin and F. Le Bars. An Interval Approach for Stability Analysis; Application to Sailboat Robotics. *IEEE Transaction on Robotics*, 27(5), 2012.

[8] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[9] H. Klinck, R. Stelzer, K. Jafarmadar, and D. K. Mellinger. AAS Endurance: An Autonomous Acoustic Sailboat for Marine Mammal Research. In *2th International Robotic Sailing Conference*, Matosinhos, Portugal, 2009.

[10] P. H. Miller, M. Hamlet, and J. Rossman. Continuous improvements to USNA sailbots for inshore racing. In *5th International Robotic Sailing Conference*, pages 49–60, Cardiff, Wales, England, 2012. Springer.

[11] T. Neumann and A. Schlaefer. Feasibility of basic visual navigation for small sailboats. In *5th International Robotic Sailing Conference*, pages 13–22, Cardiff, Wales, England, 2012. Springer.

[12] C. Petres, M. Romero Ramirez, and F. Plumet. Reactive Path Planning for Autonomous Sailboat. In *IEEE International Conference on Advanced Robotics*, pages 1–6, 2011.

[13] S. Schmitt, F. Le Bars, L. Jaulin, and T. Latzel. Obstacle Avoidance for an Autonomous Marine Robot - A Vector Field Approach. In *7th International Robotic Sailing Conference*, Irland, 2014. Springer.

[14] R. Stelzer, T. Proll, and R. John. Fuzzy Logic Control System for Autonomous Sailboats. In *in Proceedings of IEEE International Conference on Fuzzy Systems*, London, UK, 2007.